CIRRELT

Centre interuniversitaire de recherche
sur les réseaux d'entreprise, la logistique et le transport

Interuniversity Research Centre
on Enterprise Networks, Logistics and Transportation

A Heuristic Method for Non-Homogeneous Redundancy Optimization of Series-Parallel Multi-State Systems

Mohamed Ouzineb
Mustapha Nourelfath
Michel Gendreau

February 2009

CIRRELT-2009-06

Bureaux de Montréal :
Université de Montréal
C.P. 6128, succ. Centre-ville
Montréal (Québec)
Canada H3C 3J7
Téléphone : 514 343-7575
Télécopie : 514 343-7121

Bureaux de Québec :
Université Laval
Pavillon Palasis-Prince, local 2642
Québec (Québec)
Canada G1K 7P4
Téléphone : 418 656-2073
Télécopie : 418 656-2624

www.cirrelt.ca

Université Laval · UQÀM · HEC Montréal · École Polytechnique Montréal · Université de Montréal

A Heuristic Method for Non-Homogeneous Redundancy Optimization of Series-Parallel Multi-State Systems

Mohamed Ouzineb[1,2], Mustapha Nourelfath[1,3,*], Michel Gendreau[1,2]

[1] Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation (CIRRELT)

[2] Department of Computer Science and Operations Research, Université de Montréal, P.O. Box 6128, Station Centre-ville, Montréal, Canada H3C 3J7

[3] Département de génie mécanique, Pavillon Adrien-Pouliot, Université Laval, Québec, Canada G1K 7P4

Abstract. This paper develops an efficient heuristic to solve the *non-homogeneous* redundancy allocation problem for multi-state series-parallel systems. Non identical components can be used in parallel to improve the system availability by providing redundancy in subsystems. Multiple component choices are available for each subsystem. The components are binary and chosen from a list of products available on the market, and are characterized in terms of their cost, performance and availability. The objective is to determine the minimal-cost series-parallel system structure subject to a multi-state availability constraint. System availability is represented by a multi-state availability function, which extends the binary-state availability. This function is defined as the ability to satisfy consumer demand that is represented as a piecewise cumulative load curve. A fast procedure is used, based on universal generating function, to evaluate the multi-state system availability. The proposed heuristic approach is based on a combination of space partitioning, genetic algorithms (GA) and tabu search (TS). After dividing the search space into a set of disjoint subsets, this approach uses GA to select the subspaces, and applies TS to each selected subspace. The design problem, solved in this study, has been previously analyzed using GA. Numerical results for the test problems from previous research are reported, and larger test problems are randomly generated. These results show that the proposed approach is efficient both in terms of both of solution quality and computational time, as compared to existing approaches.

Keywords. Non-homogeneous redundancy optimization, multi-state, series-parallel systems, meta-heuristics.

Acknowledgements. The authors would like to thank anonymous reviewers for their comments and helpful questions. They would also like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for financial support.

Results and views expressed in this publication are the sole responsibility of the authors and do not necessarily reflect those of CIRRELT.
Les résultats et opinions contenus dans cette publication ne reflètent pas nécessairement la position du CIRRELT et n'engagent pas sa responsabilité.

---

* Corresponding author: Mustapha.Nourelfath@cirrelt.ca

This document is also published as Publication #1332 by the Department of Computer Science and Operations Research of the Université de Montréal.

Dépôt légal – Bibliothèque et Archives nationales du Québec,
Bibliothèque et Archives Canada, 2009

© Copyright Ouzineb, Nourelfath, Gendreau and CIRRELT, 2009

1 Introduction

The redundancy allocation problem (RAP) involves selection of components and levels of redundancy to maximize system performance. The RAP is NP-hard (Chern, 1992). It has attracted considerable attention from the research community. The great majority of the existing papers on the RAP use traditional binary-state reliability. It is assumed in binary-state reliability modeling that a system and its components may experience only two possible states: good and failed. The RAP for binary-state series-parallel systems has been studied in many different forms, and by considering numerous approaches and techniques. It has been solved by using optimization approaches and techniques such as dynamic programming, integer programming, mixed-integer non-linear programming, heuristics and meta-heuristics: see for example (Tillman et al., 1997; Kuo and Prasad, 2000; Kuo et al., 2001) for an extensive overview of these techniques.

However, in many real-life situations, this binary-state representation may not be adequate, since the system and its components may rather have more than two levels of performance varying from perfect functioning to complete failure. A multi-state system (MSS) may perform at different intermediate states between working perfectly and total failure. A series-parallel system consisting of different binary-state components that have a cumulative effect on the entire system performance may be considered as a MSS (Lisnianski and Levitin, 2003).

The basic concepts of MSS reliability were first introduced in (Murchland, 1975; Barlow and Wu, 1978; El-Neweihi et al., 1978; Ross, 1979). These works defined the system structure function and its properties. They also introduced the notions of minimal cut set and minimal path set in MSS context, and studied the notions of coherence and component relevancy. A literature review on MSS reliability can be found for example in (Lisnianski and Levitin, 2003). The methods currently used for MSS reliability estimation are generally based on four different approaches: (i) the structure function approach which extends Boolean models to the multi-valued case (e.g., (Barlow and Wu, 1978; El-Neweihi et al., 1978; Ross, 1979)); (ii) the Monte-Carlo simulation technique (e.g., (Zio and Podofillini, 2004)); (iii) the Markov process approach (e.g., (Xue and Yang, 1995; Pham et al., 1997)); and (iv) the universal moment generating function (UMGF) method (e.g., (Ushakov, 1986; Levitin, 2005)). These approaches are often used by practitioners, for example in the field of power systems reliability analysis (Billinton and Allan, 1990; Lisnianski and Levitin, 2003). In practice, different reliability measures can be considered for MSS evaluation and design (Aven, 1993; Liu and Kapur, 2006). The availability of a repairable MSS is defined by the system ability to meet a customer's demand (required performance level). In power systems for example, it is the ability to provide an adequate supply of electrical energy (Billinton and Allan, 1990).

The RAP for series-parallel MSS is more recent than that of binary-state systems, and it has been much less studied in the literature. It was first introduced in (Ushakov, 1987), where the universal generating function method was used for the reliability calculation (Ushakov, 1986). Following these works, genetic algorithms (GA) were used for the homogeneous RAP of series-parallel MSS in (Lisnianski et al., 1996; Levitin et al., 1998), and extended to the non-homogeneous version of this problem in (Levitin et al., 1997). The other existing solution methods are limited to the homogeneous case, and they include an ant colony optimization in (Nourelfath et al., 2003), heuristic algorithms in (Ramirez and Coit, 2004; Agarwal and Gupta, 2007), and a tabu search (TS) approach is (Ouzineb et al., 2008).

Our TS approach in (Ouzineb et al., 2008) was shown to be efficient when dealing with the

RAP of homogenous series-parallel multi-state systems. This approach proceeds by dividing the search space into a set of disjoint subsets, and then by applying TS to each subset. However, when allowing different versions of the components to be allocated in the same subsystem, the number of subsets is too high that the heuristic in (Ouzineb et al., 2008) is not able to give good quality solution in a reasonable time. The present paper extends this heuristic to solve the *non-homogeneous* case by adding a GA that selects a limited number of spaces. By applying TS only to the subspaces selected by GA (instead of the high number of possible subspaces), the proposed approach, when tested on problems from previous research and on larger problems randomly generated, proved efficient not only at reducing computing time but also at improving overall solution quality. This hybrid approach is called space partitioning/tabu-genetic (SP/TG), SP and TG being acronyms of Space Partitioning and Tabu-Genetic, respectively. It has been successfully applied in (Ouzineb et al., 2009) to solve the RAP for binary-state systems and the expansion-scheduling problem of homogeneous series-parallel multi-state systems.

The *non-homogeneous* RAP is important for two reasons (Levitin et al., 1997): (i) by allowing different versions of the components to be allocated in the same subsystem, one can obtain a solution that provides the desired availability level with a lower cost than in the solution with identical parallel components; (ii) in practice, the designer often has to include additional components in the existing system. It may be necessary, for example, to modernize a power system according to new demand levels or according to new reliability requirements. Some subsystems can contain components of versions unavailable at present. In this case, some components with the same functionality but with different parameters should compose the subsystems.

The remainder of the paper is organized as follows. In Section 2, we present a description of the RAP for *non-homogeneous* series-parallel multi-state systems. In Section 3, the solution approach is presented and applied to solve our problem. The test problems and the numerical results are presented in Section 4. Finally, some concluding remarks are given in Section 5.

2 Problem formulation

2.1 General description

The non-homogenous version of the RAP as defined in (Levitin et al., 1997), considers a system consisting of n subsystems connected in series, such that each subsystem i ($i = 1, 2, ..., n$) can contain a number of different components connected in parallel. Each subsystem has m_i versions of the component types available on the market. For each version j ($j = 1, 2, ..., m_i$) belonging to subsystem i, there are x_{ij} components connected in parallel (x_{ij} is a decision variable). Different versions and number of components may be chosen for any given subsystem. Failed components are repaired and the components availabilities are known. Each component i is characterized, according to its version j, by its availability (A_{ij}), capacity (G_{ij}) and cost (C_{ij}). The capacity is measured as a percentage of nominal total system capacity. The MSS availability is defined as its ability to satisfy consumer demand, which is represented as a piecewise cumulative load curve. Indeed, redundancy allows availability improvement, but it increases the total investment cost. The objective is to design the system so that the total component procurement cost is minimized, subject to a multi-state system availability constraint. The later is based on prescribed demand levels for different operating time periods, and it is taken to be greater than or equal to the required availability level A_0.

2.2 Assumptions

1. The characteristics of each component (*i.e.*, its cost, availability and capacity) are known and deterministic.
2. The states of the components are binary (*i.e.*, either good or failed).
3. The component states are mutually *s*-independent, and the system has a finite number of states that are *s*-independent.
4. Mixing of components is allowed in each subsystem (*i.e.*, *non-homogeneous* redundancy can be used).
5. There exists a steady-state distribution of MSS state probabilities.

2.3 Notation

n	number of series MSS subsystems
i	index for subsystem, $i \in \{1, 2, ..., n\}$
m_i	number of available components choices for subsystem i
j	index (type) for component
x_{ij}	number of components of type j used in subsystem i
$\mathbf{x}_i$	$(x_{i1}, x_{i2}, ..., x_{im_i})$
$\mathbf{X}$	a string of dimension $L = \sum_{i=1}^{n} m_i$ which defines the entire system structure, $\mathbf{X} = (x_{11}, x_{12}, ..., x_{1m_1}, x_{21}, x_{22}, ..., x_{2m_2}, ..., x_{n1}, x_{n2}, ..., x_{nm_n})$
M_{ij}	maximum number of components of version j in parallel belonging to subsystem i
A_{ij}	binary-state availability of component of version j belonging to subsystem i
C_{ij}	cost of each component of version j in subsystem i
G_{ij}	nominal performance level of component of version j in subsystem i
$C(\mathbf{X})$	total cost of series-parallel MSS
$A(\mathbf{X})$	stationary availability index of the overall multi-state series-parallel system
A_0	a specified minimum required level of system availability index
K	number of partitioned intervals
k	index for partitioned intervals
T	MSS operation period
T_k	a partitioned interval in T, $T = \sum_{k=1}^{K} T_k$
$\mathbf{T}$	vector $(T_k)_{1 \le k \le K}$
W_k^0	required MSS performance level for T_k
$\mathbf{W}^0$	vector $(W_k^0)_{1 \le k \le K}$
R_{ms}	MSS reliability
A_{ms}	MSS stationary availability
W_S	total capacity of the system
$y(t)$	MSS state at time t, $y(t) \in \{1, 2, ..., M\}$, 1 is the worst state and M is the best state

m	number or index of MSS state, $m \in \{1, 2, ..., M\}$, 1 is the worst state and M is the best state
W_m	MSS steady-state performance level associated with m
$W(t)$	output performance level of the MSS at time t, $W(t) \in \{W_1, ..., W_M\}$
p_m	$\lim_{t\to\infty}[Pr(W(t) = W_m)]$
$mnli$	maximum number of local iterations without improvement
q	amplification parameter in the penalized objective function
N_s	number of randomly-constructed solutions in the initial population of GA
N_c	number of genetic cycles
N_{rep}	number of reproduction-selection procedures per genetic cycle
CPU	the running time, given in seconds (time of execution)

2.4 Mathematical model

We first formulate the cost of subsystem as a linear function and then give the mathematical formulation of the problem. The cost of subsystem i is generally given by $\sum_{j=1}^{m_i} x_{ij}C_{ij}$. As in (Lisnianski et al., 1996; Ouzineb et al., 2008), if one has to take into account price discounting, the component cost should be considered as a function of the number of components purchased simultaneously. In this case, the cost of subsystem i is a function of x_{ij}:

$$C(\mathbf{x}_i) = \sum_{j=1}^{m_i} x_{ij}C_{ij}(x_{ij}). \tag{1}$$

The total cost can be calculated by the sum of the costs of the chosen components. The *non-homogeneous* redundancy allocation problem studied in this paper can then be stated as follows:

$$\text{minimize } C(\mathbf{X}) = \sum_{i=1}^{n} \sum_{j=1}^{m_i} x_{ij}C_{ij}(x_{ij}) \tag{2}$$

subject to

$$A(\mathbf{X}) \geq A_0, \tag{3}$$

$$x_{ij} \in \{0, 1, ..., M_{ij}\}, \quad i = 1, 2, ..., n; j = 1, 2, ..., m_i. \tag{4}$$

Constraint (3) represents the availability constraint. Constraint (4) specifies that, for each version j belonging to subsystem i, the number of components connected in parallel is an integer which cannot be higher than a pre-selected maximum number M_{ij}.

2.5 Availability of MSS

The series-parallel multi-state system is considered to have a range of performance levels from perfect functioning to complete failure. In fact, the system failure can lead to decreased capability to accomplish a given task, but not to complete failure. For example, in electric power systems, reliability is considered as a measure of the ability of the system to meet the load demand (W^0). The reliability index was defined in (Ross, 1993; Murchland, 1975). As in (Billinton and Allan, 1990), using an analogy with the Loss of Load Probability index (LOLP) we can write LOLP

$= 1 - A(X)$. LOLP is understood as the probability that the system cannot supply a given demand load. For repairable MSS, a multi-state stationary (steady-state) availability A is used as $Pr(W(t) \geq W^0)$ after enough time has passed for this probability to become constant (the reader is referred for example to (Lisnianski et al., 1996; Levitin et al., 1998)). In the steady-state the distribution of states probabilities is given by equation (5), while the multi-state stationary availability according to the demand W^0 is formulated by equation (6):

$$p_m = \lim_{t \to \infty} Pr(W(t) = W_m), \tag{5}$$

$$A(\mathbf{X}, W^0) = \sum_{W_m \geq W^0} p_m. \tag{6}$$

Since the demand is represented as a piecewise cumulative load curve, the operation period T is divided into K intervals (number of piecewises). Each interval has a duration T_k and a required demand level W_k^0 $(k = 1, ..., K)$. In this case, the MSS availability index is (Lisnianski et al., 1996; Levitin et al., 1998):

$$A(\mathbf{X}, W^0) = \frac{1}{\sum_{k=1}^{K} T_k} \sum_{k=1}^{K} A(X, W_k^0) T_k. \tag{7}$$

The equation (7) can be written as follows:

$$A(\mathbf{X}, W^0) = \frac{1}{\sum_{k=1}^{K} T_k} \sum_{k=1}^{K} Pr(W_S \geq W_k^0) T_k, \tag{8}$$

where $Pr(W_S \geq W_k^0)$ is the probability that the total capacity of the system (W_S) is not lower than a specific demand level W_k^0.

For solving the combinatorial optimization problem formulated in Section 2.4, it is important to have an effective and fast procedure to evaluate the availability of a series parallel MSS. For this evaluation, we use the universal moment generating function (UMGF), which has been proven to be very effective for high-dimensional combinatorial optimization problems. The reader is referred for example to (Lisnianski and Levitin, 2003) or (Ouzineb et al., 2008) for details about the availability evaluation method by the UMGF.

3 Solution methodology

The search space S is composed of all possible series-parallel structures. This space is first divided into a set of disjoint subspaces. Then, an efficient TS is applied to each subspace selected by using GA.

3.1 Partition of the search space

Each subspace (also called region) is characterized by its own address. Given a system structure as a string of integers $\mathbf{X}$, ($\mathbf{X} = (x_{11}, ..., x_{1m_1}, ..., x_{n1}, ..., x_{nm_n})$), the address of $\mathbf{X}$ is defined by the number of all available components. That is, address($\mathbf{X}$) $= \sum_{i=1}^{n} \sum_{j=1}^{m_i} x_{ij}$. A search subspace of address r, denoted by S_r is defined as the set of series-parallel structures (solutions) which have the same address, equal to r. While the lower bound of r is n, its upper bound is given by $N = \sum_{i=1}^{n} \sum_{j=1}^{m_i} M_{ij}$. Remark that $(S_r)_{2^s \leq r \leq N}$ is a partition of the search space S.

3.2 Tabu search

Tabu search is a meta-heuristic method originally proposed in (Glover, 1977, 1986; Glover and Laguna, 1997), that consists in an iterative, where at each iteration we move from a current solution to a new solution in a neighbourhood, until some stopping criterion has been satisfied. Our neighbourhood structure is defined as follows: at each iteration of TS, the local transformations (or moves), that can be applied to the current solution $\mathbf{X}$, define a set of neighbouring solutions for a given subspace as: Neighborhood($\mathbf{X}$) = series-parallel structures obtained by applying a single move to $\mathbf{X}$. The move applied to $\mathbf{X}$ consists in changing the number of components in parallel by adding and subtracting one, if possible, for any subsystem The move applied to $\mathbf{X}$ consists in changing the number of components in parallel by adding and subtracting one, if possible, for any subsystem ($x_{i_1 j_1} \rightarrow x_{i_1 j_1} + 1$ and $x_{i_2 j_2} \rightarrow x_{i_2 j_2} - 1$). It follows that address $\mathbf{X}$ does not change after a local transformation of $\mathbf{X}$ and *the search process remains in the same subspace*. TS enhances the local search performance by using memory structures: once a potential solution has been determined, it is marked as *tabu*, so that the algorithm does not visit that possibility repeatedly. That is, tabus are used to prevent from cycling when moving away from local optima through non-improving moves. At each iteration, the best solution $\mathbf{X}$' in a subset $V(\mathbf{X})$ Neighborhood($\mathbf{X}$) is selected and considered as a tabu solution for some next iterations. The subset $V(\mathbf{X})$, called the effective neighborhood), is generated by eliminating the tabu solutions from Neighborhood($\mathbf{X}$). Tabu solutions are stored in a short-term memory, called tabu list, which contains the solutions that have been visited in the recent past. The size of the tabu list (tabu tenure) used in this paper is dynamic, as it is usually found that using a variable size tabu list is more efficient (Gendreau, 2002; Ouzineb et al., 2008). Furthermore, the objective function (to be minimized) is penalized by adding to $C(\mathbf{X})$ the term $\alpha \max(0, A_0 - A(\mathbf{X}))$, with α is a sufficiently large number. This term represents a penalty for constraint violation, which forces the TS algorithm to consider only feasible solutions. Finally, the stopping criterion is specified in terms of a maximum number of local iterations (mnli) without improving the best-known solution.

3.3 Steady-state genetic algorithm

The GA is a meta-heuristic that operates with "chromosomal" representation of solutions, where crossover, mutation, and selection operators are applied. Instead of using only one candidate solution, a GA maintains a population of candidate solutions during its search. In our "chro-

mosomal" representation, each individual in the population is coded as a finite-length string of integers, and it represents one region in the search space. The version of GA used in this paper is commonly known as the steady-state GA or GENITOR (Whitley and Kauth, 1988). We first generate randomly an initial population of Ns solutions (strings). Within this population, new solutions are obtained during the genetic cycle by using crossover operators and applying tabu search. The crossover produces a new solution (child or offspring) from a randomly selected pair of parent solutions. We use the so-called two-point (or fragment) crossover operator, which creates the child string O for the given pair of parent strings $X1$ and $X2$ by:

1. choosing randomly two positions k and m in the string O;
2. copying string elements belonging to the fragment between k and m from $X1$; and
3. copying the rest of string elements from $X2$.

The following example illustrates this crossover procedure (the elements belonging to the fragment are in bold):

- Parent string $X1$:(2 0 0 1 3 0 0 0 0 1 1 0);
- Parent string $X2$:(**0 0 2 0 1 0 1 0 0 0 4 0**);
- Child string O:(2 0 **2 0 1 0 1 0 0** 1 1 0).

Each child solution represents a new selected subspace to which the tabu search algorithm described above is applied. To avoid the application of TS to the same subspace more than once, a set E (initially empty) is used to store the addresses of already visited subspaces, while TS is not applied to the child if its address belongs to E. Each new solution obtained by TS is decoded to obtain the objective function (fitness) values. These values, which are a measure of quality, are used to compare different solutions. This fitness is evaluated according to the penalized objective function $C(\mathbf{X}) + \alpha \max(0, A_0 - A(\mathbf{X}))$. The comparison is performed by a selection procedure that specifies which solution is better: the new solution obtained by TS or the worst solution in the population. The better solution (representing the better subspace) joins the population, while the other is discarded. If the population contains equivalent subspaces following selection, redundancies are eliminated and the population size decreases consequently. After new solutions are reproduced N_{rep} times (or if the population contains only one subspace), new randomly constructed solutions are generated to replenish the population, and a new genetic cycle begins. The GA is terminated after N_c genetic cycles. The final GA population contains the best selected subspace represented by its best solution found by TS.

4 Computational results

All the algorithms were implemented in C^{++}. The numerical tests were completed on an Intel Pentium IV 3000 MHz DEC station 5000/240 with 1024 Mbytes of RAM running under Linux. All computations use real float point precision without truncating or rounding values.

4.1 Test problems

Four design optimization problems (benchmarks) are used to analyze the SP/TG methodology for the non-homogeneous RAP for series-parallel MSS. The first three problems were introduced in (Lisnianski et al., 1996)-(Levitin et al., 1998), while the fourth is a larger instance introduced in (Ouzineb et al., 2008). For the tables of input data, the reader is referred to (Ouzineb et al., 2008) where these four benchmarks were denoted by lis4-(7/11)-4, lev5-(4/9)-4, lev4-(4/6)-3 and ouz6-(4/11)-4. In the notation *xxxa*-(*b*/*c*)-*d*, *xxx* are the first three characters of the first author's name in the paper where the instance was first introduced; a is the number of subsystems connected in series; (b/c) means that (from b to c) different components types are available in the market; and d is the number of levels in the cumulative load-demand curve. When considering that mixing of components is allowed, the total number of different solutions to be examined and the number of subspaces are given by the following equations:

$$\text{Size of the search space} = \prod_{i=1}^{n} \prod_{j=1}^{m_i} M_{ij} \tag{9}$$

$$\text{Number of subspaces} = \sum_{i=1}^{n} \sum_{j=1}^{m_i} M_{ij} - n + 1 \tag{10}$$

Let us consider for example that the maximum number of components allowed to reside in parallel is 10 for each subsystem. In this case, Table 1 presents, for each benchmark, the size of the search space and the number of subspaces for the homogeneous and non-homogeneous cases. From this table, we remark that the sizes of the problems increase drastically when considering non-homogeneous redundancy. As a result, solution methodologies for homogeneous redundancy, including our heuristic developed in (Ouzineb et al., 2008), are no longer valid.

	Cases	lev4-(4/6)-3	lev5-(4/9)-4	lis4-(7/11)-4	ouz6-(4/11)-4
Size of the search space	homogeneous	10^6	10^8	10^7	10^{10}
	non-homogeneous	10^{20}	10^{26}	10^{34}	10^{42}
Nbr. of subspaces	homogeneous	60	79	74	102
	non-homogeneous	197	286	337	415

Table 1: Search space size and disjoint subspaces (maximum of 10 components in parallel)

4.2 Parameter settings

Meta-heuristics generally depend on their parameters. Setting these parameters is an important task that can be time-consuming. Regarding this, we would like to underline first that even if the proposed heuristic combines two meta-heuristics, it has the advantage of using only a few parameters that correspond to the termination criterion. Moreover, we have performed our parameters in such a way that we have not to update their values whenever we change the problem data. In fact, preliminary numerical tests were used to set the values of the parameters. For each problem instance, other data are randomly generated and used to calibrate the parameters. Once the values of the parameters are set for these preliminary problems data, they are used

for the variations of the problem instances to be solved in this paper. In this way, we avoid any parameter's over-fitting. Parameters' values are presented in Table 2. The value of α is set to 10000 for all problems.

	lev4-(4/6)-3	lev5-(4/9)-4	lis4-(7/11)-4	ouz6-(4/11)-4
N_{rep}	100	200	300	200
N_s	500	500	500	500
mnli	20	10	20	10
N_c	10	50	50	50

Table 2: SP/TG parameters values

4.3 GA re-implementation for comparison

To the best of our knowledge, the only existing method to solve the *non-homogeneous* RAP for MSS is the GA proposed in (Levitin et al., 1997). To compare the proposed methodology with this GA, we have re-implemented it in the same conditions (computer, programming language, operating systems, etc.) as for the SP/TG heuristic. On the one hand, this allows us to determine the best feasible GA solutions for the problems that are solved for the homogeneous case in (Lisnianski et al., 1996; Levitin et al., 1998). On the other hand, it will be possible to solve by GA the problem instance in (Ouzineb et al., 2008), which is a larger benchmark also never solved in the literature for the non-homogeneous case. The values of the GA parameters are presented in Table 3. In this table, the notation P_m refers to the mutation probability used in (Levitin et al., 1997). Note that for the non-homogeneous problem solved in (Levitin et al., 1997), all the parameters are the same as in this reference. Furthermore, the parameters used for the problems newly solved by GA are calibrated using preliminary numerical tests. These parameters were varied to establish the values most beneficial to the optimization process.

	lev4-(4/6)-3	lev5-(4/9)-4	lis4-(7/11)-4	ouz6-(4/11)-4
P_m	1	1	1	1
N_s	500	500	500	500
N_{rep}	1000	2000	2000	2000
N_c	100	200	200	500

Table 3: GA parameters values

4.4 Comparisons of the proposed SP/TG with GA

The percent that one solution improves upon another is defined in terms of objective function and CPU time as:

$$\text{MPCI} = 100\% \times \frac{\text{(Minimal GA Cost - Minimal SP/TG Cost)}}{\text{Minimal GA Cost}}. \tag{11}$$

$$\text{MPTI} = 100\% \times \frac{\text{(Minimal GA Time - Minimal SP/TG Time)}}{\text{Minimal GA Time}}. \tag{12}$$

4.4.1 *Comparing the best solutions and running times over ten runs*

Using different random seeds, ten runs were made for each problem instance. The results of the best solutions obtained over ten runs by SP/TG and GA, for each instance, are presented in Tables 4 and 5. In these tables, each sub-system structure i is represented by an ordered sequence of strings as follows: $1(x_{i1}), 2(x_{i2}), ..., m_i(x_{im_i})$, that are directly taken from the corresponding solution $\mathbf{X} = (x_{11}, x_{12}, ..., x_{1m_1}, x_{21}, x_{22}, ..., x_{2m_2}, ..., x_{n1}, x_{n2}, ..., x_{nm_n})$.

Problem name	A_0	$A(\mathbf{X})$	$C(\mathbf{X})$	Best solution					
				1	2	3	4	5	6
lev4-(4/6)-3	0.900	0.901	5.423	4(1)	3(2)	1(3)	3(1), 5(1)	–	–
	0.960	0.963	7.009	1(3)	2(1), 3(2)	1(3)	3(1), 5(1)	–	–
	0.990	0.991	8.180	1(3)	3(3)	1(3)	3(1), 4(2)	–	–
lev5-(4/9)-4	0.975	0.976	12.855	4(2), 6(1)	5(6)	1(1), 4(1)	7(3)	4(3)	–
	0.980	0.981	14.770	4(2), 6(1)	3(2)	2(1), 3(2)	7(3)	3(2), 4(1)	–
	0.990	0.992	15.870	4(2), 6(1)	3(2)	2(2), 3(1)	7(3)	4(3)	
lis4-(7/11)-4	0.910	0.914	14.886	11(1)	7(1)	2(4)	3(5)	–	–
	0.920	0.920	15.075	10(1)	7(1)	2(4)	3(5)	–	–
	0.940	0.941	17.418	1(5)	7(1)	2(5)	2(1), 3(4)	–	–
	0.950	0.950	19.861	1(5)	3(2)	2(5)	2(3), 3(2)	–	–
	0.960	0.961	20.570	10(1)	2(1), 5(2)	2(4)	2(1), 3(4)		
	0.970	0.970	21.288	10(1)	2(1), 5(2)	2(5)	2(3), 3(2)	–	–
	0.980	0.981	22.738	1(5)	2(1), 5(2)	2(5)	2(1), 3(4)	–	–
	0.990	0.990	23.779	1(5)	1(1), 5(2)	2(5)	2(5)	–	–
	0.999	0.999	26.919	1(6)	3(4)	2(5)	2(3), 3(3)	–	–
ouz6-(4/11)-4	0.975	0.979	11.241	3(4)	1(4)	2(5)	2(7)	3(2)	4(1)
	0.980	0.980	11.369	3(4)	1(5)	2(5)	2(8)	3(2)	4(1)
	0.990	0.992	12.764	3(4)	1(4)	2(4)	2(8)	3(2)	4(2)

Table 4: SP/TG best solutions over ten runs

Problem name	A_0	$A(\mathbf{X})$	$C(\mathbf{X})$	Best solution					
				1	2	3	4	5	6
lev4-(4/6)-3	0.900	0.900	5.803	2(1), 3(1)	3(2)	1(3)	3(1), 4(1)	–	–
	0.960	0.960	7.365	2(2)	2(2), 3(1)	1(3)	2(2), 3(1)	–	–
	0.990	0.991	8.180	1(3)	3(3)	1(3)	3(1), 4(2)	–	–
lev5-(4/9)-4	0.975	0.976	12.855	4(2), 6(1)	5(6)	1(1), 4(1)	7(3)	4(3)	–
	0.980	0.981	14.770	4(2), 6(1)	3(2)	2(1), 3(2)	7(3)	3(2), 4(1)	–
	0.990	0.992	15.870	4(2), 6(1)	3(2)	2(2), 3(1)	7(3)	4(3)	–
lis4-(7/11)-4	0.910	0.914	14.886	11(1)	7(1)	2(4)	3(5)	–	–
	0.920	0.920	15.075	10(1)	7(1)	2(4)	3(5)	–	–
	0.940	0.941	17.418	1(5)	7(1)	2(5)	2(1), 3(4)		
	0.950	0.950	19.861	1(5)	3(2)	2(5)	2(3), 3(2)	–	–
	0.960	0.961	20.570	10(1)	2(1), 5(2)	2(4)	2(1), 3(4)	–	–
	0.970	0.970	21.288	10(1)	2(1), 5(2)	2(5)	2(3), 3(2)	–	–
	0.980	0.981	22.562	10(1)	3(3)	2(5)	2(4), 3(1)	–	–

Problem name	A_0	$A(\mathbf{X})$	$C(\mathbf{X})$	Best solution					
				1	2	3	4	5	6
	0.990	0.990	23.779	1(5)	1(1), 5(2)	2(5)	2(5)		
	0.999	0.999	26.919	1(6)	3(4)	2(5)	2(3), 3(3)		
	0.975	0.979	11.241	3(4)	1(4)	2(5)	2(7)	3(2)	4(1)
ouz6-(4/11)-4	0.980	0.980	11.516	3(5)	1(5)	2(5)	2(7)	3(2)	4(1)
	0.990	0.990	12.911	3(5)	1(4)	2(4)	2(7)	3(2)	4(2)

Table 5: GA best solutions over ten runs

In the comparison results given in Table 6, the best solutions and the best running times are independently selected among the ten runs. These results indicate that the proposed SP/TG generally outperforms the GA. In fact, in terms of the solution quality, Table 6 shows that SP/TG obtained lower cost in 4 of the 18 test cases, one higher cost, and it is equal to GA in the other 13 cases. In terms of the execution time, SP/TG is quicker than GA for all instances.

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
	0.900	**5.423**	5.803	06.55	**03.68**	10.03	63.31
lev4-(4/6)-3	0.960	**7.009**	7.365	04.83	**04.12**	10.29	59.96
	0.990	**8.180**	**8.180**	00.00	**03.68**	10.34	64.41
	0.975	**12.855**	**12.855**	00.00	**18.48**	71.01	73.98
lev5-(4/9)-4	0.980	**14.770**	**14.770**	00.00	**17.32**	61.05	71.63
	0.990	**15.870**	**15.870**	00.00	**21.36**	67.09	68.16
	0.910	**14.886**	**14.886**	00.00	**61.46**	166.75	63.14
	0.920	**15.075**	**15.075**	00.00	**56.05**	172.72	67.55
	0.940	**17.419**	**17.418**	00.00	**52.70**	139.48	62.22
	0.950	**19.861**	**19.861**	00.00	**25.37**	137.28	81.52
lis4-(7/11)-4	0.960	**20.570**	**20.570**	00.00	**46.33**	154.23	69.96
	0.970	**21.288**	**21.288**	00.00	**38.83**	156.68	75.22
	0.980	22.738	**22.562**	-00.77	**62.14**	112.79	44.91
	0.990	**23.779**	**23.779**	00.00	**64.69**	119.73	45.97
	0.999	**26.919**	**26.919**	00.00	**70.20**	197.44	64.44
	0.975	**11.241**	**11.241**	00.00	**77.00**	372.76	79.34
ouz6-(4/11)-4	0.980	**11.369**	11.516	01.28	**66.60**	415.45	83.97
	0.990	**12.764**	12.911	01.14	**45.58**	377.57	87.93

Table 6: The comparison results (the best costs and running times among ten runs are in bold

4.4.2 Comparing the mean values over ten runs

The mean values obtained by each method (*i.e.*, mean costs and mean CPU over ten runs) are given in Table 7. In terms of the mean objective function, SP/TG obtained lower costs in 13 of the 18 test cases, obtained higher costs in 2 of the 18 test cases, and is equal to GA in the other 3 cases. In terms of the mean running time, SP/TG is quicker than GA for all instances.

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
lev4-(4/6)-3	0.900	**5.423**	5.803	06.55	**04.59**	10.17	54.87
	0.960	**7.009**	7.369	04.88	**04.72**	10.36	54.44
	0.990	**8.193**	8.195	00.02	**04.47**	10.45	57.22
lev5-(4/9)-4	0.975	**12.855**	**12.855**	00.00	**28.68**	74.14	61.32
	0.980	**14.774**	14.778	00.01	**25.51**	69.44	63.26
	0.990	**15.870**	**15.870**	00.00	**25.66**	74.36	65.49
lis4-(7/11)-4	0.910	**14.886**	14.937	00.34	**125.16**	183.17	31.67
	0.920	**15.075**	**15.075**	00.00	**81.51**	210.64	61.30
	0.940	**17.477**	17.500	00.13	**142.27**	174.00	18.24
	0.950	**19.861**	20.363	02.46	**73.87**	180.73	59.13
	0.960	**20.570**	20.802	01.12	**108.99**	174.80	37.65
	0.970	**21.324**	21.424	00.47	**72.71**	194.24	62.57
	0.980	**22.738**	22.781	00.19	**106.31**	194.26	45.27
	0.990	**23.825**	23.867	00.18	**147.81**	151.08	2.16
	0.999	27.391	**26.923**	-01.71	**122.62**	209.41	41.45
ouz6-(4/11)-4	0.975	11.296	**11.241**	-00.49	**101.54**	382.71	73.47
	0.980	**11.401**	11.516	00.10	**96.22**	419.77	77.08
	0.990	**12.779**	12.911	01.02	**96.49**	388.37	75.16

Table 7: The comparison results in terms of the mean values (ten trials)

4.4.3 *Comparing the worst results over ten runs*

The worst results obtained by each method (*i.e.*, worst costs and worst CPU among the ten runs) are given in Table 8. In terms of the worst objective function, SP/TG obtained lower costs in 8 of the 18 test cases, obtained higher costs in 6 of the 18 test cases, and is equal to GA in the other 4 cases. In terms of the worst running time, SP/TG is quicker than GA in 14 of the 18 test cases and it is less efficient in the other 4 cases.

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
lev4-(4/6)-3	0.900	**5.423**	5.803	06.55	**05.82**	10.34	43.71
	0.960	**7.009**	7.403	05.32	**05.84**	10.53	44.54
	0.990	**8.315**	8.328	00.16	**05.66**	10.58	46.56
lev5-(4/9)-4	0.975	**12.855**	**12.855**	00.00	**38.68**	77.64	50.18
	0.980	14.789	**14.780**	-00.06	**33.68**	71.24	52.72
	0.990	**15.870**	**15.870**	00.00	**31.46**	77.96	59.56
lis4-(7/11)-4	0.910	**14.886**	15.218	02.18	406.95	**194.66**	-52.17
	0.920	**15.075**	**15.075**	00.00	**140.52**	225.04	37.56
	0.940	18.004	**17.711**	-01.63	375.92	**199.73**	-46.87
	0.950	**19.861**	20.748	04.27	**174.07**	226.18	23.04
	0.960	**20.570**	21.148	02.73	234.27	**195.37**	-16.60
	0.970	21.556	**21.541**	-00.01	**134.30**	218.10	38.42

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
	0.980	**22.738**	22.965	00.99	**174.81**	260.10	32.79
	0.990	**24.136**	24.227	00.38	424.76	**172.94**	-59.28
	0.999	29.249	**26.952**	-07.58	**211.19**	228.90	07.74
	0.975	11.794	**11.241**	-04.69	**143.74**	389.33	63.08
ouz6-(4/11)-4	0.980	11.687	**11.516**	-01.46	**142.78**	432.91	67.02
	0.990	**12.911**	**12.911**	00.00	**158.64**	394.81	59.82

Table 8: The comparison results in terms of the worst values (among ten trials)

4.4.4 Comparing the average results of SP/TG with the best results of GA

From Table 9, we conclude that if the average solutions of SP/TG are compared to the best solutions of GA, in 4 instances SP/TG is better, in 8 instances GA is better, and in the remaining instances (6) they are equal. Moreover, when comparing the average running time of SP/TG with the best CPU of GA, in 16 instances SP/TG is better, and in 2 instances GA is better. This comparison of average performances of SP/TG versus best-of-ten GA performances (GA) highlights clearly the efficiency of SP/TG.

Problem name	A_0	Cost			Running time		
		SP/TG	*GA*	*MPCI*	*SP/TG*	*GA*	*MPTI*
	0.900	**5.423**	5.803	06.55	**04.59**	10.03	54.24
lev4-(4/6)-3	0.960	**7.009**	7.365	04.83	**04.72**	10.29	54.13
	0.990	8.193	**8.180**	-00.16	**04.47**	10.34	56.77
	0.975	**12.855**	**12.855**	00.00	**28.68**	71.01	59.61
lev5-(4/9)-4	0.980	14.774	**14.770**	-00.01	**25.51**	61.05	58.21
	0.990	**15.870**	**15.870**	00.00	**25.66**	67.09	61.75
	0.910	**14.886**	**14.886**	00.00	**125.16**	166.75	24.94
	0.920	**15.075**	**15.075**	00.00	**81.51**	172.72	52.80
	0.940	17.477	**17.418**	-00.34	142.27	**139.48**	-01.96
	0.950	**19.861**	**19.861**	00.00	**73.87**	137.28	46.19
lis4-(7/11)-4	0.960	**20.570**	**20.570**	00.00	**108.99**	154.23	29.33
	0.970	21.324	**21.288**	-00.17	**72.71**	156.68	53.59
	0.980	22.738	**22.562**	-00.77	**106.31**	112.79	05.74
	0.990	23.825	**23.779**	-00.19	147.81	**119.73**	-19.00
	0.999	27.391	**26.919**	-01.72	**122.62**	197.44	37.89
	0.975	11.296	**11.241**	-00.49	**101.54**	372.76	72.76
ouz6-(4/11)-4	0.980	**11.401**	11.516	01.00	**96.22**	415.45	76.84
	0.990	**12.779**	12.911	01.02	**96.49**	377.57	74.44

Table 9: Comparison of mean SP/TG with best GA performances over 10 seeds

4.5 Robustness

To measure the robustness of the SP/TG and GA algorithms, the standard deviations over ten runs are given in Table 10 for each instance. We remark from this table that for each instance, the standard deviation is very low for both algorithms. This implies that the proposed method and GA are both robust. Nevertheless, our SP/TG has a lower standard deviation in 13 of the 18 test cases. Therefore, SP/TG can be considered as more robust than GA, while yielding generally solutions with lower costs in a shorter time.

Problem name	A_0	SP/TG	GA
lev4-(4/6)-3	0.900	00.00	00.00
	0.960	00.00	00.02
	0.990	00.04	00.05
lev5-(4/9)-4	0.975	00.00	00.00
	0.980	00.01	00.00
	0.990	00.00	00.00
lis4-(7/11)-4	0.910	00.00	00.11
	0.920	00.00	00.00
	0.940	00.18	00.11
	0.950	00.00	00.34
	0.960	00.00	00.15
	0.970	00.09	00.13
	0.980	00.00	00.20
	0.990	00.11	00.19
	0.999	00.97	00.01
ouz6-(4/11)-4	0.975	00.17	00.00
	0.980	00.10	00.00
	0.990	00.05	00.00

Table 10: Standard deviations of SP/TG and GA

4.6 Convergence of SP/TG and GA

The convergence curves of SP/TG and GA were drawn for all the 18 test cases, showing similar behaviour in all these cases. Figures 1-4 present examples of these curves and show that the proposed SP/TG algorithm requires generally less numbers of iterations than GA. We remark that even if the solutions of our algorithm are generally not as good as GA solutions during the first iterations of the search process, they tend to become quickly better.

5 Conclusion

The non-homogeneous redundancy allocation problem for multi-state series-parallel systems has been efficiently solved in this paper by combining two meta-heuristics, and the idea of space portioning that proceeds by dividing the search space into a set of disjoint regions. A steady-state

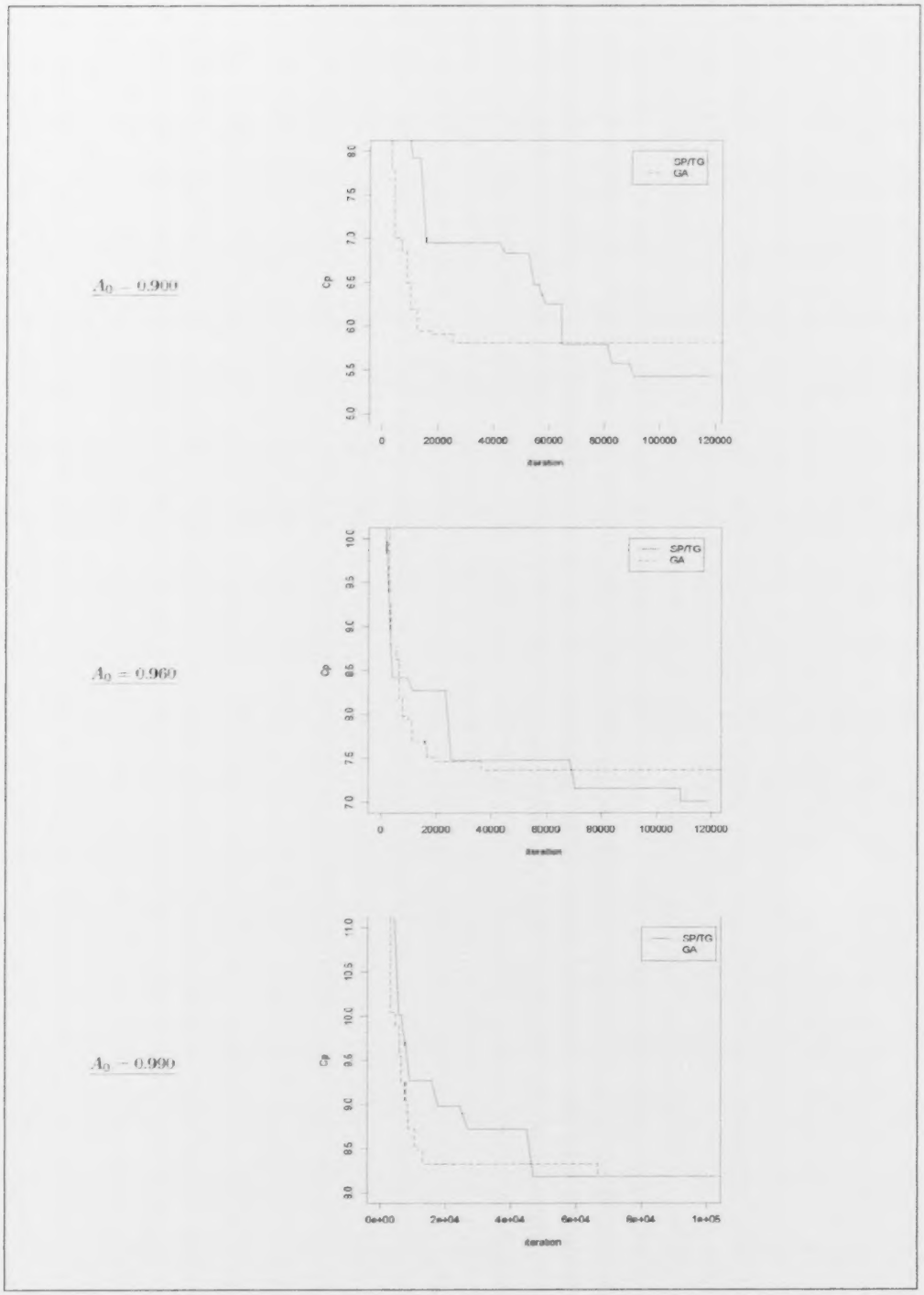


Figure 1: Convergence curves for GA and SP/TG for lev4-(4/6)-3

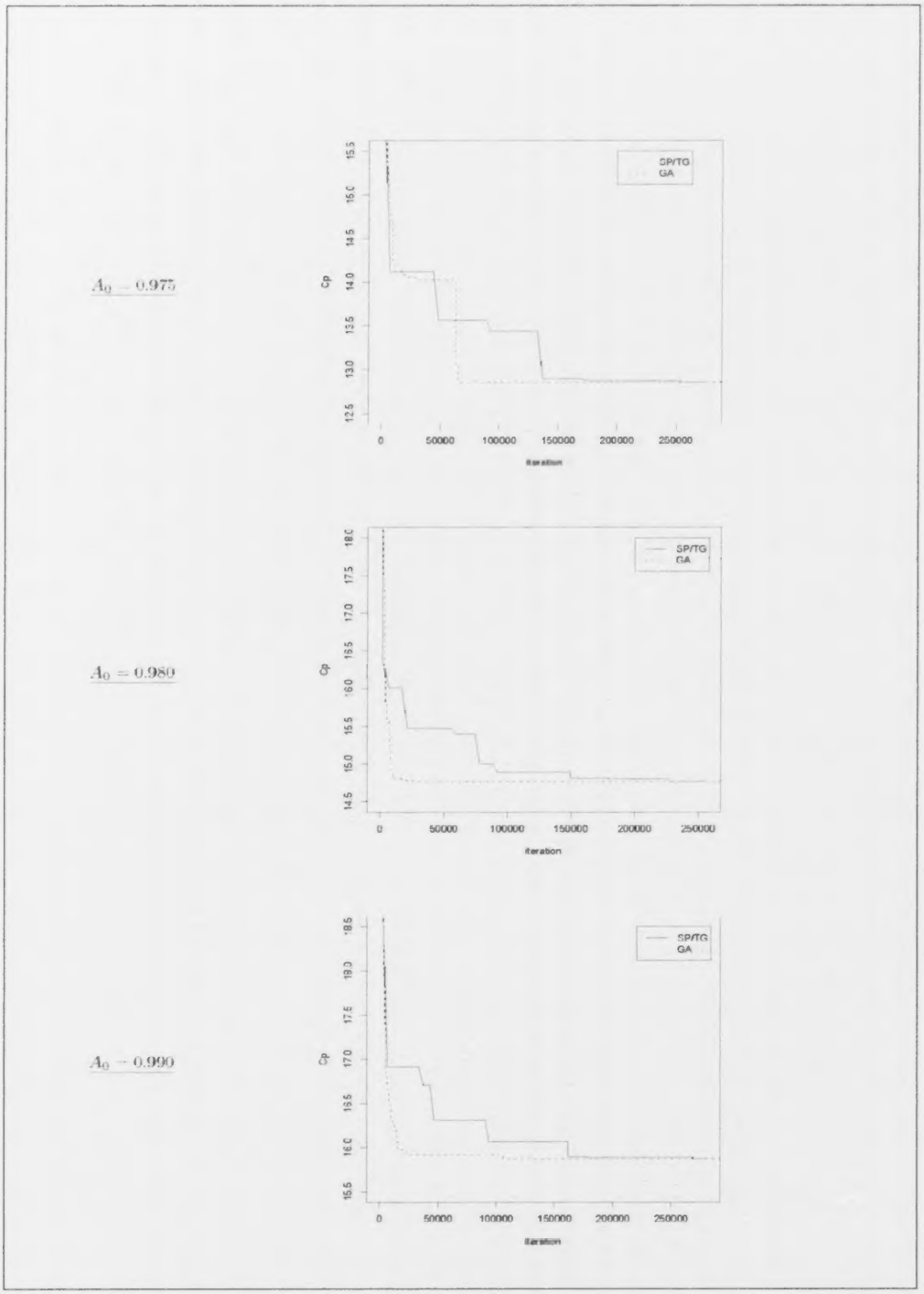


Figure 2: Convergence curves for GA and SP/TG for lev5-(4/9)-4

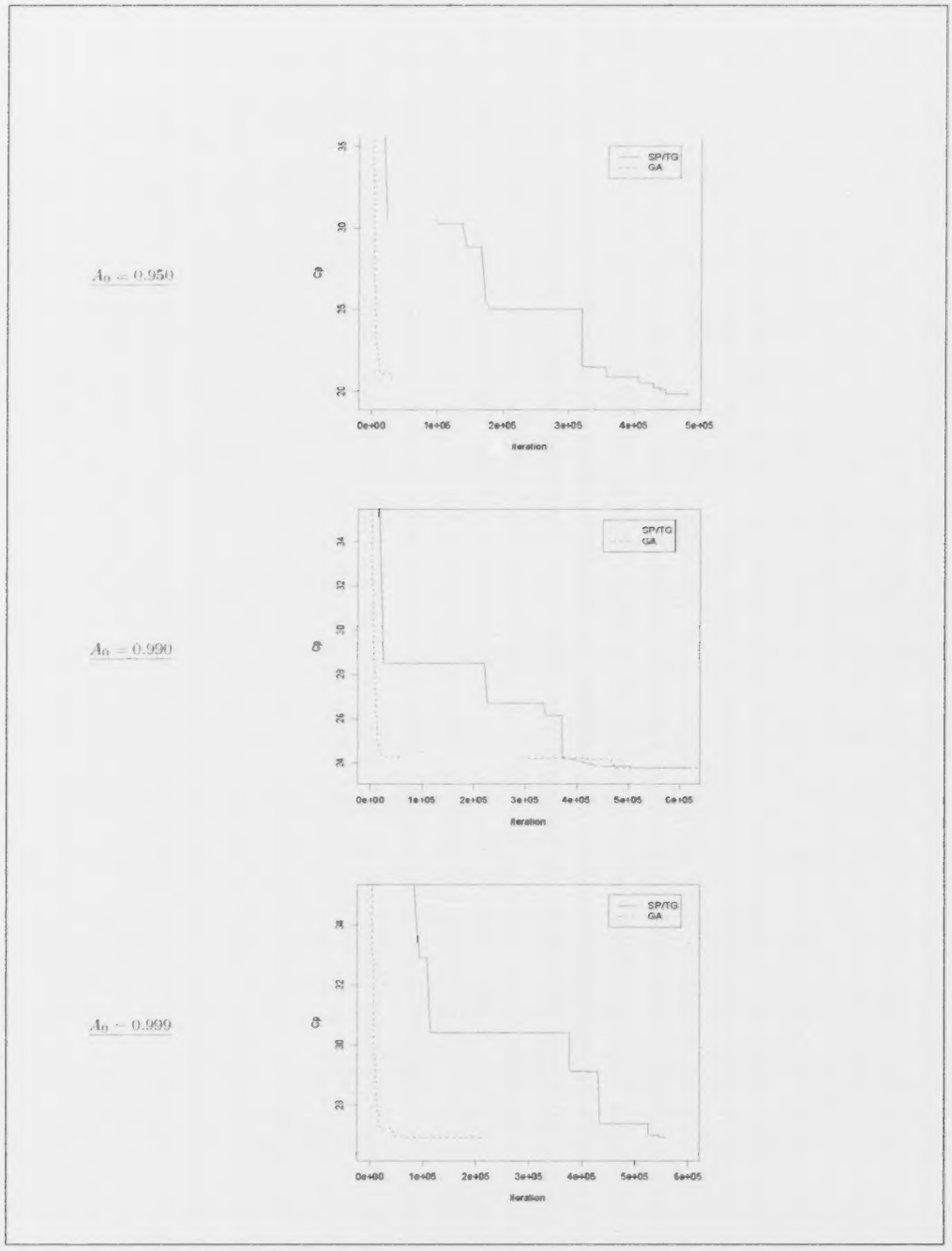
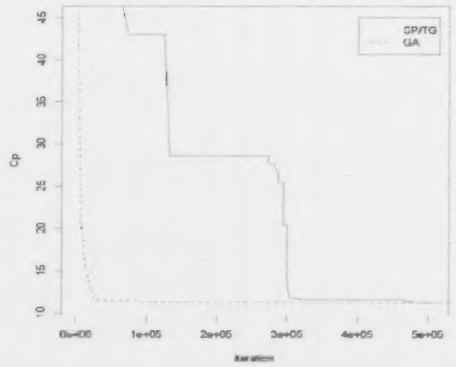
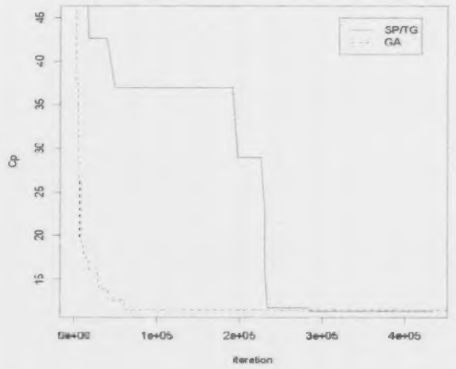


Figure 3: Convergence curves for GA and SP/TG for lis4-(7/11)-4

$A_0 = 0.975$



$A_0 = 0.980$



$A_0 = 0.990$

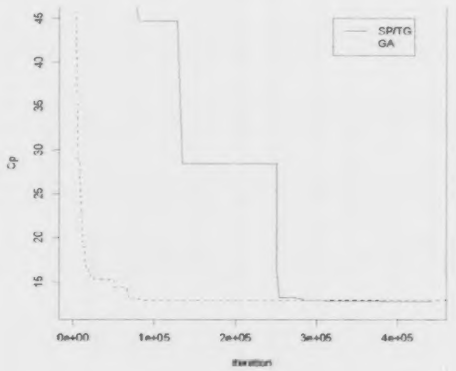


Figure 4: Convergence curves for GA and SP/TG for ouz6-(4/11)-4

genetic algorithm has been used for the selection of subspaces, while an efficient tabu search has been applied to each selected subspace. Four test problems have been analyzed and the results have been attractive, showing that the proposed methodology can be used as an efficient alternative to GA. When compared to GA, SP/TG results in a superior performance in terms of solution quality, execution time and reduced variability. By re-implementing the GA, we have not only shown the greater efficiency of our SP/TG but also re-proven that the GA proposed in (Levitin et al., 1997) is effective for the non-homogeneous RAP of series-parallel MSS.

References

Agarwal, M. and Gupta, R. (2007) Homogeneous redundancy optimization in multi-state series-parallel systems: A heuristic approach. *IIE Transactions, 39(3)*, 277–289.

Aven, T. (1993) On performance measures for multistate monotone systems. *Reliability Engineering and System Safety, 41*, 259–266.

Barlow, S.E. and Wu, A.S. (1978) Coherent systems with multi-state components. *Mathematics of Operations Research, 3(11)*, 275–281.

Billinton, R. and Allan, R. (1990) *Reliability evaluation of power systems.* Pitman.

Chern, M.S. (1992) On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letters, 11*, 309–315.

El-Neweihi, E., Proschan, F. and Sethuraman, J. (1978) Multistate coherent systems. *Journal of Applied Probability, 15(12)*, 675-688.

Gendreau, M. (2002) *Recent Advances in Tabu Search.* in Essays and Surveys in Metaheuristics, C.C. Ribeiro and P.Hansen(eds.). Kluwer Academic Publishers, 369-377.'

Glover, F. (1977) Heuristics for Integer Programming Using Surrogate Constraints. *Decision Sciences, 6*, 156-166.

Glover, F. (1986) Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 13*, 533-549.

Glover, F. and Laguna, M. *Tabu Search.* Kluwer Academic Publishers, Boston 1997.

Kuo, W. and Prasad, VR. (2000) An Annotated Overview of System-reliability Optimization. *IEEE Trans on Reliab, 49(2)*, 176–187.

Kuo, W., Prasad, VR., Tillman, F.A. and Hwang C.L. (2001) *Optimal Reliability Design: Fundamentals and applications.* Cambridge University Press.

Levitin, G., Lisnianski, A., Ben-Haim, H. and Elmakis, D. (1997) Structure optimization of power system with different redundant elements. *Electric Power Syst Res, 43(1)*, 19-27.

Levitin, G., Lisnianski, A., Ben-Haim, H. and Elmakis, D. (1998) Redundancy optimization for series-parallel multi-state systems. *IEEE Trans on Reliab, 47(2)*, 165–172.

Levitin, G. (2005) *Universal generating function in reliability analysis and optimization.* Springer-Verlag.

Lisnianski, A. and Levitin, G. (2003) *Multi-state system reliability. Assesment, optimization and applications.* World scientific.

Lisnianski, A., Levitin, G., Ben-Haim, H. and Elmakis, D. (1996) Power system structure optimization subject to reliability constraints Electric. *Power Syst Res, 39(2)*, 145–152.

Liu, Y. W., and Kapur, K. C. (2006) Reliability measures for dynamic multistate nonrepairable systems and their applications to system performance evaluation, *IIE Transactions, 38(6)*, 511–520.

Murchland, J. (1975) Fundamental concepts and relations for reliability analysis of multi-state systems. Reliability and Fault Tree Analysis. *Theoretical and Applied Aspects of System Reliability, SIAM*, 581-618.

Nourelfath, M., Nahas, N. and Zeblah, A. (2003) An ant colony approach to redundancy optimization for multi-state system. *International conference on industrial engineering and production management (IEPM'2003)*, Porto-Portugal.

Ouzineb, M., Nourelfath, M. and Gendreau, M., (2008) Tabu search for the redundancy allocation problem of homogenous series-parallel multi-state systems. *Reliability Engineering and System Safety, 93(8)*, 1257–1272.

Ouzineb, M., Nourelfath, M. and Gendreau, M., (2009) An efficient heuristic for reliability design optimization problems. Submitted to *Computers and Operations Research*, May 2008.

Pham, H., Suprasad, A. and Misra, R. B. (1997) Availability and mean life time prediction of multistage degraded system with partial repairs. *Reliability Engineering and System Safety, 56*, 169–173.

Ross S. (1979) Multivalued state component systems. *Annals of Probability, 7*, 379–383.

Ramirez, M. and Coit, D.W. (2004) A heuristic for solving the redundancy allocation problem for multi-state series-parallel systems, *Reliab Eng Syst Saf, 83*, 341–349.

Ross, S. (1993) *Introduction to probability models.* Academic press.

Tillman, FA, Hwang, CL and Kuo W. (1997) Optimization Techniques for System Reliability with Redundancy A review. *IEEE Trans on Reliab, 26(3)*, 148–155.

Ushakov, I. (1987) Optimal standby problems and a universal generating function. *Sov. J. Computing System Science, 25(4)*, 79–82.

Ushakov, I. (1986) Universal generating function. *Sov. J. Computing System Science, 24(5)*, 118–129.

Whitley, D. and Kauth, J. (1988) *GENITOR: A diffrent genetic algorithm.* Tech. ReportCS-88-101, Colorado State University.

Xue, J. and Yang, K. (1995) Dynamic reliability analysis of coherent multi-state systems, *IEEE Transactions on Reliability, 44*, 683–638.

Zio, E. and Podofillini, L. (2004) A Monte Carlo approach to the estimation of importance measures of multi-state components. *Reliability and Maintainability Annual Symposium (RAMS)*, 129–134.